Learning to Bid: Deep Counterfactual Regret Minimization in Fantasy Hockey Auctions

Sparsh Agrawal

May 2025

Abstract

This report presents a novel application of Deep Counterfactual Regret Minimization (Deep CFR) to optimize bidding strategies in fantasy hockey auction drafts. We explore the challenges of developing an AI agent capable of competing with human players in a complex, imperfect information game setting. Through iterative development and testing of multiple approaches including AlphaZero and Proximal Policy Optimization (PPO), we demonstrate that Deep CFR provides an effective framework for learning competitive bidding strategies. Our implementation successfully handles the unique constraints of fantasy hockey auctions, including fixed budgets, roster requirements, and sequential decision-making under uncertainty.

1 Introduction

1.1 Problem Statement

Fantasy hockey auction drafts present a complex optimization challenge where n players must draft teams consisting of o forwards, d defensemen, and g goalies, each working within a fixed budget b. The auction follows an English format where players either raise bids by 1 or fold, with the last remaining player winning the athlete at their bid price. This creates a rich strategic environment combining resource management, valuation assessment, and competitive bidding.

1.2 Game Environment

Our final implementation uses a custom game environment with the following key characteristics:

- Fixed budget of $100 per player
- Support for 4 players
- Pre-defined athlete pools with known values:
  - 24 forwards (values ranging from 278-520)
  - 16 defensemen (values ranging from 191-307)
  - 8 goalies (values ranging from 193-273)
- Discrete action space with 101 possible bid values (0-100%)

1.3 Existing Methods

Current fantasy sports platforms offer limited automated drafting capabilities that exhibit several exploitable weaknesses:

- **Value-Based Bidding**: Existing systems typically use simple value-based bidding where they:
  - Bid up to a fixed percentage of an athlete's projected value
  - Fail to adapt to market dynamics
  - Ignore position scarcity and roster composition
- **Predictable Patterns**: Common exploitable behaviors include:
  - Consistent early-round overbidding
  - Rigid maximum bid thresholds
  - Failure to capitalize on late-round value
- **Market Opportunity**: The limitations of current systems create opportunities for:
  - More sophisticated bidding strategies
  - Dynamic adaptation to opponent behavior
  - Improved resource allocation across positions

1.4 Motivation

The project emerged from personal experience with fantasy hockey auctions, where the lack of sophisticated automated strategies presented an opportunity for innovation. While platforms like ESPN offer basic automated drafting, these systems employ simplistic strategies that experienced players can easily exploit. Often if someone doesn't show up to the draft, an automated bot will play for them, and this bot ends up being very exploitable. This gap in the market, combined with the theoretical richness of the problem, motivated our research into developing more robust bidding strategies.

2 Literature Review

2.1 Current State of the Field

The domain of fantasy sports auction optimization remains largely unexplored in academic literature. Current industry solutions, such as ESPN's automated drafting system, rely on analyst-suggested values with hard-coded bidding caps, making them predictable and exploitable. This presents an opportunity for applying modern machine learning techniques to develop more sophisticated strategies.

2.2 Related Work

While direct precedent is limited, our work draws from several related fields:

- Colonel Blotto games, which share similar resource allocation dynamics
- Sequential auction theory, providing theoretical foundations
- Deep learning applications in imperfect information games

3 Methodology

3.1 Approach Evolution

We tried three different approaches in this project:

3.1.1 AlphaZero Method

Initially, we explored an AlphaZero-style approach using Monte Carlo Tree Search (MCTS). The policy function was defined as:

$$\pi^*(s) = \arg \max_{a \in \{\text{raise,fold}\}} Q(s, a)$$

However, the enormous search space ($O(2^{nb})$) and computational overhead made this approach impractical.

3.1.2 PPO Implementation

We then investigated Proximal Policy Optimization with a clipped surrogate objective:

$$L^{CLIP}(\theta) = \mathbb{E}_t \Big[\min \left(r_t(\theta) A_t,\ \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t \right) \Big]$$

The policy outputted a mean and variance which was moment-matched to a beta distribution:

$$\text{bid} = b_r \cdot x, \quad x \sim \text{Beta}(\alpha, \beta)$$

3.1.3 Deep CFR Final Implementation

Our final approach used Deep Counterfactual Regret Minimization, focusing on expected reward prediction:

$$Q(s, a) = \text{Expected Reward given action } a$$

With regret calculation:

$$R(s, a) = Q(s, a) - \sum_{a'} \pi(s, a') Q(s, a')$$

And policy updates via regret matching:

$$\sigma(s, a) = \frac{\max(R(s, a), 0)}{\sum_{a'} \max(R(s, a'), 0)}$$

4 Approach Comparison and Challenges

Each approach we explored presented unique challenges and insights, leading to our final choice of Deep CFR:

4.1 AlphaZero Challenges

The AlphaZero approach faced fundamental limitations that made it impractical for our use case:

- **Computational Intractability**
  - Game tree complexity of $O(2^{nb})$ made training untenable
  - Even with significant simplifications, could not scale beyond trivial cases

  - The only case that we solved was a simple case where the optimal strategy was to bid as high as you could.

- **Scaling Issues**
  - Attempts to reduce game size still hit computational barriers
  - MCTS search depth requirements grew exponentially
  - Memory requirements became prohibitive quickly

4.2 PPO Implementation Issues

The PPO approach revealed fundamental limitations in handling the auction dynamics:

- **Credit Assignment Problems**
  - Critical early-game decisions (bidding on top players) were undervalued
  - Traditional reward discounting actively hurt performance as it insufficiently credited early game actions
  - Temporal structure of auction games poorly suited to PPO framework

- **Action Distribution Modeling**
  - Beta distribution moment matching converged to point estimates
  - A limitation of this approach is also the parameterization of the strategy space as a beta distribution. A priori it is not clear that such a unimodal distribution is the best way to parameterize the strategy space. However, I will note our final implementation often plays uniomodal distributions so this need not be a limitation.
  - Alternative clipped normal distributions showed no improvement

4.3 Deep CFR Advantages and Modifications

Our final Deep CFR implementation included several key modifications to the standard algorithm:

- **Key Modifications**
  - Q-function prediction instead of direct regret estimation. My belief was this function would be more stable over time as it doesn't depend on a volatile policy
  - Linear reservoir sampling weighted by iteration number which lead to faster convergence
  - Extended to handle n-player scenarios. Even though there is no theoretical guarantee, it empirically worked well

4.3.1 Q-Function Modification

Our modification to the standard Deep CFR algorithm replaces direct regret estimation with Q-function learning:

- **Traditional CFR**
  - Directly estimates cumulative regret

    - Updates based on counterfactual value differences
    - Can be unstable in early training
  - **Q-Function Approach**
    - Learns expected value of state-action pairs
    - Regret computed from Q-values: $R(s,a) = Q(s,a) - \sum_{a'} \pi(s,a')Q(s,a')$
    - More stable training dynamics
    - Better generalization across similar states

4.3.2 Linear Reservoir Sampling

Our custom reservoir sampling approach weights samples by their iteration:

- **Implementation**
  - Sample weight proportional to iteration number
  - Newer samples more likely to be retained
  - Balances exploration and exploitation
- **Benefits**
  - Better adaptation to evolved strategies
  - Improved sample efficiency
  - Natural curriculum learning effect

4.3.3 N-Player Extension

Extending Deep CFR to n-player scenarios required several considerations:

- **State Space Handling**
  - Expanded state representation for multiple players
  - Efficient encoding of multi-player history
  - Scalable memory management
- **Training Dynamics**
  - Increased variance in value estimates
  - More complex equilibrium landscape
  - Longer convergence times

- **Theoretical Benefits**
  - Strong game theoretical guarantees from CFR framework
  - Natural handling of imperfect information
  - Better suited to sequential decision-making
- **Implementation Advantages**
  - Q-function approach provided more stable training
  - Linear reservoir sampling improved sample efficiency
  - Successfully scaled to multi-player scenarios

4.4 Comparative Analysis

Our exploration of these approaches revealed several key insights:

- **Computational Feasibility**
  - AlphaZero: Intractable beyond trivial cases
  - PPO: Computationally feasible but strategically limited
  - Deep CFR: Best balance of computational cost and performance
- **Strategic Depth**
  - AlphaZero: Potentially optimal but couldn't scale
  - PPO: Limited by action distribution modeling
  - Deep CFR: Rich strategy space with theoretical backing
- **Practical Applicability**
  - AlphaZero: Limited to minimal test cases
  - PPO: Struggled with crucial early-game decisions
  - Deep CFR: Successfully handled full game complexity

5 Technical Implementation

5.1 State Representation

The state space for our fantasy hockey auction environment consists of several key components:

- **Player Information**
  - Current budget (normalized to [0,1])
  - Roster slots filled by position
  - Historical bidding patterns
- **Auction State**
  - Current athlete being auctioned
  - Current bid amount
  - Active bidders
  - Round number
- **Global Information** There are many other features that would be relevant such as:
  - Remaining athletes by position
  - Average value of remaining athletes
  - Position scarcity metrics

  However, because we train on a fixed universe of athletes with a fixed nomination order, these are embedded in the game's structure.

5.2 Custom Sampling Strategy

Our implementation uses a novel sampling approach to improve training stability:

```
def custom_weight_calculation(self, sample_num):
    # Exponential weighting scheme
    base = np.random.uniform()
    exponent = 100000 / sample_num #constant for numerical stability
    weight = base ** exponent
    return weight / self.total_weight
```

This approach provides several benefits:

- Balances exploration and exploitation
- Reduces variance in early training
- Improves convergence stability

5.3 Batching Optimizations

We implemented three levels of simulation batching that provided significant performance improvements:

5.3.1 Sequential Simulation

Basic implementation with no parallelization, serving as our baseline:

```
for env in environments:
    action = model.predict(env.state)
    next_state, reward = env.step(action)
```

5.3.2 Partial Batching

Intermediate optimization with batched predictions, which achieved a 5-7x speedup over sequential simulation:

```
states = torch.stack([env.state for env in environments])
actions = model.predict_batch(states)
results = [env.step(a) for env, a in zip(environments, actions)]
```

This approach batches the neural network inference across multiple environments, significantly reducing GPU overhead by processing multiple states simultaneously.

5.3.3 Full Batching

Advanced implementation with branch merging, which achieved an additional 3-4x speedup over partial batching (15-20x total speedup over sequential):

```
class BatchedSimulator:
    def __init__(self, batch_size=2048):
        self.batch_size = batch_size
        self.active_envs = []

    def step_batch(self):
        states = self.collect_states()
        actions = self.model.predict_batch(states)
        self.process_actions(actions)
        self.merge_similar_branches()
```

The full batching approach extends batching to all aspects of simulation:

- Batches neural network inference across all active environments
- Merges similar decision branches to reduce redundant computation
- Processes multiple game trajectories in parallel
- Efficiently handles branch termination and new branch creation

This optimization was crucial for making training feasible, reducing simulation time for 1000 games from approximately 2 hours to 6 minutes. The dramatic speedup enabled us to scale the game beyond the 2 player version into a 4 player game.

5.4 Memory Management

To handle the large state space efficiently, we implemented several memory optimization techniques:

- **Reservoir Sampling**: Maintains fixed-size buffers for training data
- **State Compression**: Efficient encoding of game states. Locking in the player universe and nomination order allowed for a much more condensed state space.

6 Implementation

6.1 Neural Network Architecture

We implemented two key neural networks:

6.1.1 Regret Network

The regret network predicts counterfactual values for state-action pairs:

```
    class RegretNetwork(nn.Module):
    def __init__(self, hidden_sizes=[128, 64, 32]):
        super().__init__()
        input_size = FUNCTION_CONFIG["obs_dim"] + 1
        layers = [nn.Linear(input_size, hidden_sizes[0]), nn.ReLU()]

        for i in range(len(hidden_sizes) - 1):
            layers.append(nn.Linear(hidden_sizes[i], hidden_sizes[i+1]))
            layers.append(nn.ReLU())

        self.network = nn.Sequential(*layers)
        self.output_layer = nn.Linear(hidden_sizes[-1], 1)

    def forward(self, x):
        x = self.network(x)
        output = self.output_layer(x)
        return torch.tanh(output).squeeze(-1
```

6.1.2 Policy Network

The policy network learns the optimal bidding strategy:

```
    class PolicyNetwork(nn.Module):
    def __init__(self, hidden_sizes=[256, 256, 128]):
        super().__init__()
        input_size = FUNCTION_CONFIG["obs_dim"]
        layers = [nn.Linear(input_size, hidden_sizes[0]), nn.ReLU()]

        for i in range(len(hidden_sizes) - 1):
            layers.append(nn.Linear(hidden_sizes[i], hidden_sizes[i+1]))
            layers.append(nn.ReLU())

        self.network = nn.Sequential(*layers)
        self.output_layer = nn.Linear(hidden_sizes[-1], FUNCTION_CONFIG["n_discrete"])

    def forward(self, x):
        x = self.network(x)
        log_probs = torch.log_softmax(self.output_layer(x), dim=-1)  # Log-
    probabilities
        return log_probs
```

6.2 Training Process

Our training implementation includes several key optimizations:

6.2.1 Configuration Parameters

- Policy reservoir size: 25,000 samples. This is smaller as the output space is much larger.
- Q value reservoir size: 1,000,000 samples
- Batch size: 2,048
- Learning rate: 0.001
- Training iterations: 10,000

6.3 Loss Functions

6.3.1 Policy Network Training

We use Jensen-Shannon divergence for policy network training. This was chosen as KL divergence was unstable as it has unbounded loss values. Jensen-Shannon divergence is a symmetric divergence that is always non-negative and bounded.

```
def jensen_shannon_divergence(log_p, q, eps=1e-12):
    p = torch.exp(log_p)
    q = q + eps
    q = q / q.sum(dim=-1, keepdim=True)
    m = 0.5 * (p + q)
    log_m = torch.log(m + eps)
    kl_pm = F.kl_div(log_p, m, reduction="batchmean")
    kl_qm = F.kl_div(log_m, q, reduction="batchmean")
    return 0.5 * (kl_pm + kl_qm)
```

6.3.2 Q Function Network Training

For the Q function network, we use weighted mean squared error:

$$\text{Loss} = \mathbb{E}_{s,a,r,w\sim\mathcal{D}}\left[w \cdot (Q(s,a) - r)^2\right]$$

where w is the importance weight calculated as:

$$w = \frac{1}{\text{reach} + \epsilon} \cdot \frac{1}{\sum_i w_i}$$

7 Results

7.1 Qualitative Performance

Our Deep CFR implementation demonstrated strong performance across various testing scenarios:

- **Human Expert Evaluation**
  - Successfully competed against course professor
  - Performed well against experienced fantasy hockey players
  - Demonstrated non-trivial bidding strategies that surprised human experts
- **Multi-Player Dynamics**
  - Particularly strong performance in two-player scenarios
  - Showed competent but less dominant performance in four-player games
  - Suggests interesting dynamics between game complexity and human comprehension
- **Strategic Depth**
  - Developed sophisticated budget management strategies
  - Demonstrated understanding of position scarcity
  - Adapted bidding behavior based on opponent actions. For example, in the two player version, often times the agent would bid such that it would ensure it still had more capital than you for later players.

7.2 Training Infrastructure

The system was trained on a 2020 MacBook Pro with the following characteristics:

- **Training Duration**
  - 566 iterations
  - Total training time of several days. Each iteration takes roughly 10 minutes.
  - Efficient memory utilization throughout training.
- **Scaling Characteristics**
  - Two-player scenarios showed faster convergence
  - Four-player scenarios required more training time
  - Suggests potential benefits from more extensive training or model complexity

7.3 System Adaptability

The implementation demonstrates strong adaptability across different scenarios:

- **League Configuration**
  - Easily adaptable to different scoring systems. We jsut adjust our expected player values accordingly.
  - Flexible reward function design. Currently the agent optimizes for win probability and rank, this could be adjusted to optimize for other metrics based off your league's payout structure.
  - Configurable for various roster requirements. We could adjust the number of players and the number of roster spots to be more or less than the standard 12 player roster with 6 forwards, 4 defensemen, and 2 goalies.

7.4 Key Observations

Several important insights emerged from our testing:

- Two-player scenarios may actually present more complex strategy spaces than initially assumed. Because your actions affect your opponents more there is more complexity involved in developing optimal counter-strategies. As the number of players n increases, having more optimal player valuations starts to matter more than the actual auction strategy.
- Human players found it harder to develop optimal counter-strategies in two-player games
- System's performance scaled differently with number of players than expected
- Suggests interesting directions for future research in multi-player dynamics

8 Future Work

8.1 Performance Improvements

Several promising directions for improvement include:

- **Scaling To More Complex Settings**
  - Larger model architectures for complex multi-player dynamics
  - Specialized training regimes for >2 players
  - Improved opponent modeling
  - Would like to see greater generalizability in the player universe. Ideally a single model that can adapt to different player pools and scoring systems would be developed. We were unable to acomplish that in this project.
  - Removing the assumption of consensus on player valuations would be interesting. In the real world there is disagreement on how much players are going to score in a season, modelling that in would be a great step forward.

8.2 Extensions to Other Domains

The techniques developed here could be applied to other auction settings:

- **General Auction Domains**
  - Real estate auctions
  - Online marketplace bidding
  - Resource allocation markets

8.3 Practical Improvements

Several practical enhancements could make the system more useful:

- **User Interface**
  - Web-based interface for easier access
  - Real-time strategy visualization
  - Interactive training process monitoring
- **Integration Features**
  - API for fantasy sports platforms
  - Real-time data updates
  - Custom scoring system support
- **Analysis Tools**
  - Strategy explanation features
  - Post-draft analysis
  - Training progress visualization

8.4 Theoretical Extensions

Several theoretical directions warrant further investigation:

- **Equilibrium Analysis**
  - Formal proof of convergence in n-player setting
  - Characterization of equilibrium properties
  - Impact of information structure on strategies
- **Algorithm Improvements**
  - Alternative Q-function architectures
  - Novel sampling strategies
  - Hybrid approaches combining multiple methods

9 Practical Implications

9.1 Lessons from Implementation

Our implementation revealed several practical insights:

- **Algorithm Selection**
  - Theoretical guarantees often translate to practical benefits
  - Simpler approaches (AlphaZero) can fail due to scaling issues
  - Balance between theoretical elegance and practical feasibility is crucial
- **Training Considerations**
  - Consumer hardware can be sufficient for meaningful results
  - Careful algorithm modification more important than computational power
  - Efficient implementation crucial for practical training times

9.2 Real-World Applications

The project has several immediate practical applications:

- **Fantasy Sports**
  - Direct application to fantasy hockey auctions
  - Adaptable to other fantasy sports formats
  - Potential for integration with existing platforms
- **Training Tool**
  - Helps players understand optimal bidding strategies
  - Reveals counter-intuitive strategic insights
  - Provides practice environment for human players

9.3 Key Insights

Several important insights emerged from our work:

- **Strategic Complexity**
  - Two-player scenarios often more complex than anticipated
  - Early-game decisions disproportionately important
  - Position scarcity creates interesting strategic dynamics
- **Algorithm Design**
  - Q-function estimation more stable than direct regret learning
  - Sample weighting crucial for efficient learning
  - Multi-player scenarios require careful handling

9.4 Limitations

Important limitations to consider include:

- **Computational**
  - Training time increases significantly with player count
  - Memory requirements grow with game complexity
  - Real-time performance considerations for live use
- **Strategic**
  - Performance gap between 2-player and n-player scenarios
  - Potential for unexplored strategic spaces
  - Difficulty in explaining agent decisions
- **Practical**
  - A lack of quantitative metrics to evaluate the performance of the agent. We are limited to human feedback and the results of the agent in the auction.
  - Current UI limitations
  - Need for better visualization tools
  - Integration challenges with existing platforms
  - Restrictions on a fixed universe of athletes and nomination order. This makes it difficult to apply this in practice directly. Users likely need to train a model for their own league, understand what players are usually worth and then adjust according to the auction dynamics.

10 Conclusion

This project demonstrates the successful application of Deep CFR to fantasy hockey auction drafts, creating a competitive AI agent capable of sophisticated bidding strategies. The work not only advances the state of automated fantasy sports drafting but also provides insights into the application of modern machine learning techniques to complex game environments.

11 Acknowledgments

Special thanks to Professor Damek for his guidance and support throughout this project!